组件间多种通信方式:

> 1). 组件间通信最基本方式: props
> 2). 组件间通信高级1: vue自定义事件与事件总线
> 3). 组件间通信高级2: v-model
> 4). 组件间通信高级3: .sync 属性修饰符
> 5). 组件间通信高级4: $attrs与$listeners
> 6). 组件间通信高级5: $children与$parent
> 7). 组件间通信高级6: 作用域插槽slot-scope
> 8). 组件间通信高级7: vuex

DOM原生事件与组件事件前言:

```
原生DOM事件
    绑定原生DOM事件监听的2种情况
        在html标签上绑定DOM事件名的监听
        在组件标签上绑定DOM事件名的监听，事件绑定在组件的根标签上
    当用户操作对应的界面时，浏览器就会自动创建并封闭包含相关数据的事件对象，分发对应的事件，从而触发事件监听回调函数调用
    事件对象event，本质是 "事件数据对象"
    event对象内的数据属性: target / offsetX / offsetY / keyCode等
    $event就是浏览器创建的event对象，默认传递给事件监听回调函数的就是它
```

一、关于props通信 《父向子通信，只读，单向数据流》

- **概念:**
  - 在父子组件间进行通信
  - 父可向子传递**非函数/函数数据**
  - 若传递为**非函数数据**则是将父组件的数据让其子组件可访问
  - 传递函数数据，其本质是父组件需要子组件所**分发**的数据

```
  用来实现父子之间相互通信的最基本方式，也是用得最多的方式
父 ==> 子，传递的是非函数类型的属性
  子 ==> 父，传递的是函数类型的属性
  问题: 其它关系的组件使用props就会比较麻烦
```

注意:

知识点补充：组件标签中的所有属性参数，都已经传递给了子组件，前面加上冒号只是为了动态单向数据绑定

子组件接收数据(props)的三种写法:

- 1.布尔值
- 2.对象
- 3.函数

```
//写法1
  props:{
        propA:['data','data1']
  }
  props:['data']

  //写法2 指定数据格式
  props:{
    propB:Number
  }

  //写法3 指定多个检查参数
  props:{
    propC:{
    type:Number,
        default:0
    }
  }
```

特殊情况：

路由组件之间没有标签，但是可以把参数通过路由映射为属性（当数据过多时不推荐）

关于路由配置中使用props

路由配置中的props

在vue 路由配置文件，配置了props，作用是可以与route解耦，取值方便

三种props配置：

- **1 设置为true**

```
        {
            path："/news/:type?",
            component: () => import(/* webpackChunkName: "news chunk name" */
"News组件”),
            props: true
        }
   在News组件中，使用如下：
        {
            props：["type"], // 在props指定需要的属性名称
            data: () => {},
            methods: {}
            ...
        }
```

- **2 设置为对象**

```
    {
        path: "/news/:type?",
        component: () => import(/* webpackChunkName: "news chunk name" */ "News
组件"),
        props: {
                country: 'China',
                time: '6pm'
        }
    }
在News组件中，使用如下：
    {
        props: ["country", "time"], // 在props指定需要的属性名称
        data: () => {},
        methods: {}
        ...
    }
```

- **3 设置为函数(返回对象)**

```
    {
        path: "/news/:type?",
        component: () => import(/* webpackChunkName: "news chunk name" */ "News
组件"),
        props:  (route) => ({
                    country: route.query.country,
                    time: route.query.time
                })
    }
使用方式 与 1，2 相同
```

传输流程：

- 1、于父组件引入 (import) 其子组件,并注册子组件（components）
- 2、父组件使用模板（template）定义子组件标签（子组件name），通过v-bind（**：（冒号）**）来向子组件**动态**传输数据
- 3、子组件仅需要定义（props）接收数据即可

二、Vue自定义事件:《子向父通信》

原生事件：

- 1、事件类型　　内置事件
- 2、回调函数　　自定义
- 3、调用者　　系统/浏览器(分发，触发，调用)
- 4、默认实参　　event 事件对象

自定义事件：

- 1、事件类型　　无穷大个
- 2、回调函数　　自定义
- 3、调用者　　自己调用
- 4、默认实参　　指定参数或无参数（无事件对象）

概念：

```
绑定vue自定义事件监听
    只能在组件标签上绑定
    事件名是任意的，可以与原生DOM事件名相同
只当执行$emit('自定义事件名', data)时分发自定义事件，才会触发自定义事件监听函数调用
$event：就是分发自定义事件时指定的data数据
$event可以是任意类型，甚至可以没有
```

关于$emit与$on

关键点：

- **注意判断当前事件是否为自定义事件！《组件标签中的事件为自定义事件，html标签中的事件为原生事件》**
- **当前非兄弟组件时，自定义事件定义在当前接收数据的组件对象中，若当前为全局事件总线，自定义事件定义在$bus上**
- **回调定义在接收数据的组件上，且接收数据的位置为这个$on方法上的第二个参数《回调执行时候返回给当前模板使用》**
- **发送数据的组件中的自定义事件分发定义在接收数据的组件上，依靠$emit方法的第二个参数向接收数据组件实现通信传递参数**

使用组件标签定义事件（非原生事件）时，通常情况下无法触发需要在事件名后添加**.native**属性

原生事件与组件事件的区别：

- 原生事件在html标签上是**默认带有事件对象**的（即不写入实参也会自带事件对象且可使用）
- 原生的事件在**组件标签上**则**不再带有事件对象**（即不写入实参则无参数接收，有参数则为实参数据，无事件对象）
- 原生的事件**在组件标签上就不是原生的事件了**，此时已经成为了**自定义事件**

若想要在组件标签中使用原生事件：

使用.native以后，更改为原生事件，此时事件绑定在根元素上　（当前事件源的共有父类）《事件委派》此时所有子标签都可触发该回调

```vue
<template>
  <div>
    <h1>EventTest组件</h1>
    <!-- 原生dom事件在html标签和组件标签上的区别    （Event1组件测试） -->
    <!--
      原生的事件在html标签上就是原生的
      原生的事件在组件标签上就默认不是原生的（而是自定义事件），想要成为原生事件
    -->
    <button @click="test1">我爱你赵丽颖</button>
    <!-- <button @click="test1($event)">我爱你赵丽颖</button> -->
    <Event1 @click.native="test2"></Event1>
                        更改为原生事件

  </div>
```

注意：

$on中的第一个参数为事件名，第二个参数为接收数据的实参**，在**$emit分发数据的组件中，**第一个参数为需要接收数据的时间回调名，第二个参数，则是所需要向父组件传递的参数。**

传输流程：

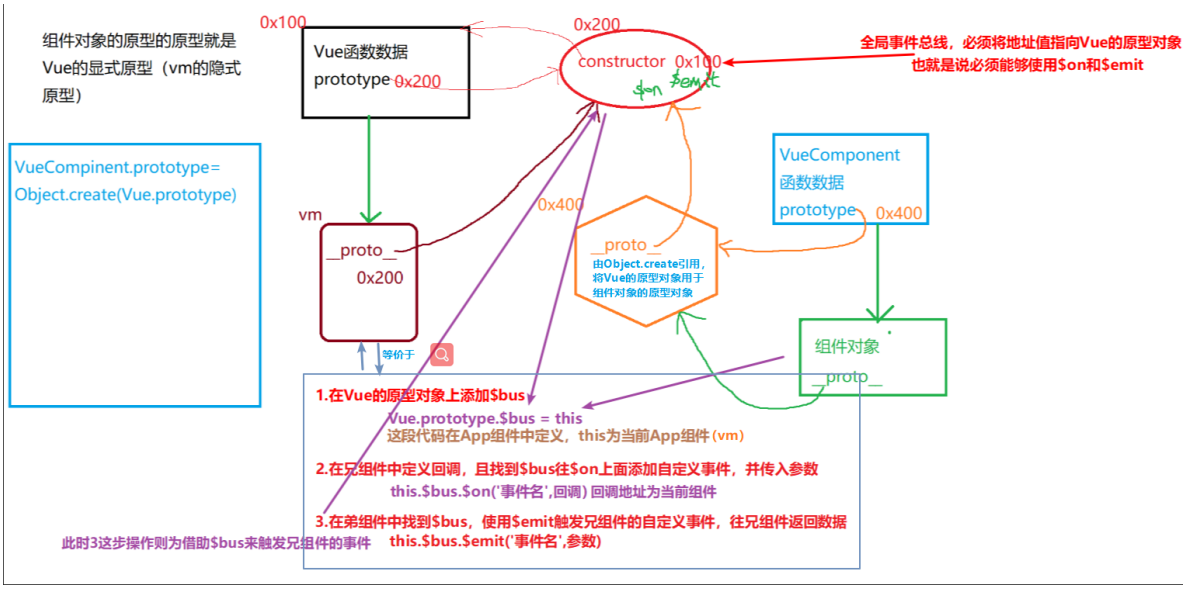
- 1、于父组件中通过components对子组件进行注册
- 2、通过template中模板标签使用，创建并引入数据 **:item="item"**
- 3、子组件通过props定义接收父组件template中所引入的数据
  - 父组件定义回调监听，子组件进行分发（触发），子组件使用$emit分发，一旦事件被调用，将数据返回给其回调定义的组件中
  - 普通标签原生事件：参数不写的情况下，默认为事件对象$events
  - 组件标签事件　　：参数不写的情况下，没有默认值

最终结论：

- **原生dom事件在html标签和组件标签上的区别：**
  - 在html标签上，添加的事件为原生dom事件
  - 在组件标签上添加的则是自定义事件，若想成为原生事件，则使用**.native**将自定义事件更改为原生事件，本质上是将其原生dom事件添加到了组件根元素上（template的直接子元素）
- **Vue自定义事件在html标签和组件标签上的区别**
  - 在html标签上添加的自定义事件无意义，所以自定义事件本质上是给组件标签添加的
  - 参数一：事件名（任意，可与原生同名），但是这最终还是一个自定义事件

三、全局事件总线《主兄弟之间通信，任意组件皆可通信》

图示：



概念：

- 该通信方式，可以让任意组件进行通信。其原理同props,$parent,ref
- 而EventBus借助中间件$bus（于Vue实例化对象的隐式原型对象）来访问其对应的组件对象上的数据
- 通信要求：
  - 1、所有组件对象均可见
    - **注意：当前Vue.prototype.$bus的地址值是否是一个组件对象，若不是，则无法访问$on和$emit**
  - 2、能够使用$on （事件监听）能够使用$emit （事件触发,分发）

  理解:

```
    Vue原型对象上有3个事件处理的方法: $on() / $emit() / $off()
    组件对象的原型对象是vm对象: 组件对象可以直接访问Vue原型对象上的方法
    实现任意组件间通信
编码实现:
    将入口js中的vm作为全局事件总线对象:
        beforeCreate() {
            Vue.prototype.$bus = this
        }
    分发事件/传递数据的组件: this.$bus.$emit('eventName', data)
    处理事件/接收数据的组件: this.$bus.$on('eventName', (data) => {})
```

传输流程：

- 1、对应要求1，所有组件均可见
  - 于Vue的原型对象上添加$bus方法《只要其他组件可见，均可定义》
- **事件与触发都绑定在$bus上，而回调定义在接收数据的（兄组件）组件上，发送数据的组件（弟组件）会触发$bus上的事件并传参**
  - 比如说，Vue的实例化对象隐式原型指向的Vue的原型对象，组件对象的隐式原型对象的隐式原型对象所指向的Vue的原型对象
- 2、在要接收数据的组件中的**自定义事件中**，于页面**挂载完毕(mounted)以后**，通过this.$bus找到该原型对象，并指定事件回调，将事件存入这个原型中而**回调定义在兄组件中**

```
this.$bus.$on="(事件名,this.回调)"
```

- 3、在发送数据的组件中的**原生事件中**，于当前组件找到$bus

```
this.$bus.$emit('事件名',this.返回的数据<相当于实参>)
```

- **并使用$emit来触发$bus上的的事件，返回数据，返回给兄组件的methods回调执行返回给当前组件的模板中去使用**
    - **注意：事件名在触发与监听方法中必须同名**

> # 四、关于V-model通信《通常实现带表单的父子组件双向数据同步问题》
>
> 封装复用性组件

概念：

```
原生input上的v-model的本质：  动态的value属性与原生input事件监听
    <input type="text" :value="name2" @input="name2=$event.target.value">
组件标签上的v-model的本质：  动态的value属性与自定义input事件监听
    父组件:
        <CustomInput :value="name4" @input="name4=$event"/>
    子组件:
        props: ['value']
        <input type="text" :value="value" @input="$emit('input',
$event.target.value)">
利用v-model能做什么?
    v-model不仅能实现原生标签与其父组件间数据双向通信(同步)
    也能实现父子组件间数据双向通信(同步)
    一般用于封装带表单项的复用性组件
    elment-ui中: Input/CheckBox/Radio/Select等表单项组件都封装了v-model
```

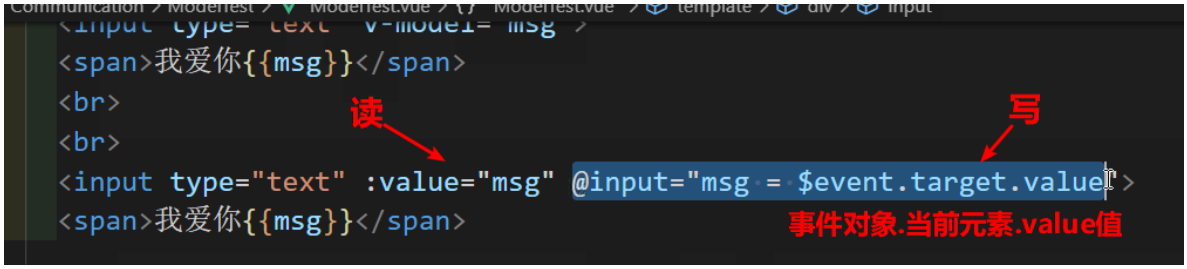
知识点补充：

```
1、html input v-model的本质
    :value = “data”  //读取数据
    @input = "data = $event.target.value"  //写数据
```

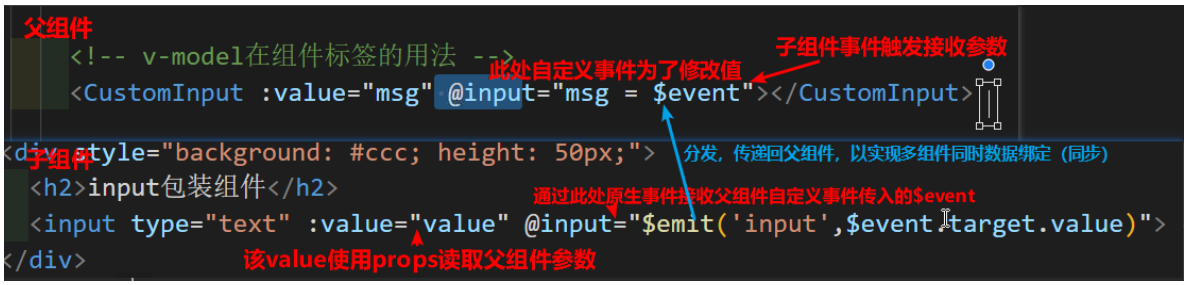
```
2、组件标签上 v-model本质
    :value = "data"  父组件传递属性给子组件，子组件需要接受
    @input = "data = $event"
    数据在父组件当中     子组件当中必须这样写
    先接收props:['value']

    子组件表单类元素
        : value = "value"
        @input = "@emit('input',$event.target.value)"
```

原生中：

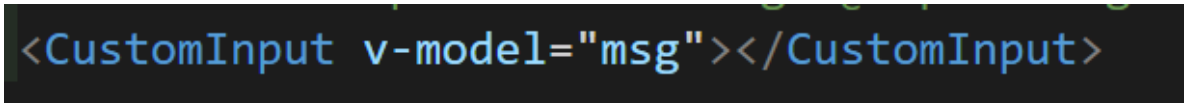
此写法为V-model的本质写法：



组件中：



《最终的写法》父组件中子组件标签读写参数的简写方式：



传输流程：

本质是将当前data中数据利用props传递给另外一个组件，而另外一个组件使用$emit()分发数据返回给当前这个组件，以实现组件间数据绑定

根据v-model本质写法特性衍生：

- 1、 **父组件中** 通过v-model的本质写法，于组件标签中创建@input自定义事件，准备接收数据
  - 读：
    - 读取发送数据的组件（**子组件**）中所返回的值，单向数据绑定（**:value="msg" 读取data中（父组件）的数据传递给子组件**）
  - 写：
    - 在**父组件**标签中**使用该@input同原生事件名的自定义事件**中定义的msg，而这个自定义事件的msg = $event，这个$event相当于是一个形参，其实参来自于（**子组件中**）触发**子组件的原生inpit事件读取当前这个子组件中所修改的value值，然后再使用$emit()分发这个组件中所修改过后的value表单内容**

```
<CustomInput :value="msg" @input="msg = $event" ></CustomInput>
```

- 2、在（**子组件**）触发，分发事件的组件中，使用props接收数据，

```
props:['value']
```

- 3、在子组件模板中定义input标签，绑定当前输入框的value，且定义emit()分发触发事件

```
<input
  type="text"
  :value="value"
  @input="$emit('input', $event.target.value)"
/>
```

五、Sync 同步《通常实现不带表单的父子组件双向数据同步问题》

实现组件的隐藏

概念：

```
理解本质：绑定一个自定义事件监听，用来接收子分组分发事件携带的最新数据来更新父组件的数据
    <child :money.sync="total"/>
    <Child :money="total" @update:money="total=$event"/>
利用sync能做什么呢?
    实现父子组件间数据双向同步
    常用于封装可复用组件(需要更新父组件数据)
        v-model一般用于带表单项的组件
        sync一般用于不带表单项标签的组件
    element-ui中：Dialog就利用sync来实现组件的隐藏
```

通信流程：《本质上还是自定义事件》

v-model 实现效果几乎一样

父组件给子组件属性传递数据后面添加.sync

子组件修改数据 需要分发事件@click = $emit("update:属性名",要更新的数据)

本质写法：

```
//父组件：定义自定义事件回调，并向子组件传入父组件的参数
<Child :money="money" @spendMoney="spendMoney"></Child>
//定义回调
  methods:{
    spendMoney(money){
      this.money -= money
    }
  },
  data(){
```

```
        return{
          money:1000
        }
    },

//子组件：使用props接收父组件传递的参数，当前点击事件触发时，分发，触发父组件自定义事件，传入参数，供父组件回调执行返回数据使用
<button @click="$emit('spendMoney',100)">花钱</button>
```

.sync的本质写法：

sync本质上也是绑定了一个事件，但是事件名是固定的，不能随意更改

```
//父组件
<Child :money="money" @update:money="spendMoney"></Child>
//定义回调
  methods:{
    spendMoney(money){
      this.money -= money
    }
  },
//初始化数据
  data(){
        return{
          money:1000
        }
    },
```

使用.sync分发，触发父组件的同步事件《同上需要使用props接收父组件传递的参数》

```
//子组件
<button @click="$emit('update:money',100)">花钱</button>
```

.sync的简化写法《此时不需要在父组件定义回调》：

```
//父组件
<Child2 :money.sync="money"></Child2>
//初始化数据
  data(){
        return{
          money:1000
        }
    },
```

分发父组件回调，《同上需要使用props接收父组件传递的参数》最终将emit中的第二个参数返回给父组件回调

```
<button @click="$emit('update:money',money - 100)">花钱</button>
```

六、$attrs和$linsteners《二次封装组件》

概念：

```
理解：
    $attrs：排除props声明，class，style的所有组件标签属性组成的对象
    $listeners：级组件标签绑定的所有自定义事件监听的对象
    v-bind：的特别使用：<div v-bind="{ id: someProp, 'other-attr': otherProp }">
</div>
    v-on：的特别使用：<button v-on="{ mousedown: doThis, mouseup: doThat }">
</button>
    一般：v-bind与$attrs配合使用，v-on与$listeners配合使用
使用它们能做什么呢?
    在封装可复用组件时：HintButton
        从父组件中接收不定数量/名称的属性或事件监听
        在组件内部，传递给它的子组件
    element-ui中：Input就使用了v-bind与$attrs来接收不定的属性传递给input
扩展双击监听：
    @dblclick="add2"
        绑定是自定义事件监听，而el-button内部并没处理(没有绑定对应的原生监听，没有分发自定义
事件)
        双击时，不会有响应
    @dblclick.native="add2"
        绑定的是原生的DOM事件监听，最终是给组件的根标签<a>绑定的原生监听
        当双击a内部的<button>能响应，因为事件有冒泡
```

本质就是父组件中给 子组件传递的所有属性组成的对象及自定义事件方法组成的对象

- **$attrs**:
  - 在子组件中，$attrs可以获取到所有父组件中，通过子组件标签中定义的所有属性参数，**但是除了props已经接收过的数据**
- **$attrs 如果不声明props 那么子组件当中是可以看到 如果声明了哪个属性，那么那个属性在$attrs当中看不到**
  - *它会排除 props声明接收的属性 以及class style**
  - 可以通过v-bind 一次性把父组件传递过来的属性添加给子组件
- 可以通过v-on 一次性把父组件传递过来的事件监听添加给子组件
- **$linsteners**:
  - 在子组件中，$linsteners**可以获取到所有父组件中，通过子组件标签中定义的所有事件和方法**

案例使用$attrs和$linsteners对element - ui 进行二次封装：

本质写法：《自定义带Hover提示的按钮》

```
    <a href="javascript:;">
      <el-button
        type="primary"
        icon="el-icon-delete"
        title="删除！！"
      ></el-button>
    </a>
```

二次封装写法：

```
//父组件中，直接在二次封装的组件标签中写入属性和事件
<template>
  <div>
    <h2>自定义带Hover提示的按钮</h2>
    <HintButton
      type="danger"
      icon="el-icon-delete"
      title="删除"
      @heihei="haha"
      @click="hahaha"
    ></HintButton>
    <HintButton
      type="warning"
      icon="el-icon-edit"
      title="修改"
      @heihei="haha"
      @click="hahaha"
    ></HintButton>
  </div>
</template>
```

可以通过v-bind 一次性把父组件传递过来的属性添加给子组件

可以通过v-on 一次性把父组件传递过来的事件监听添加给子组件

```
//二次封装组件
<template>
  <a href="javascript:;" :title="title">
    //v-bind绑定$attrs获取到父组件的所有属性，除了props中的title，因为title使用了props，
且在a标签中需要使用
    //v-on绑定了父组件$listeners获取到的所有事件方法所形成的对象
    <el-button v-bind="$attrs" v-on="$listeners"></el-button>
  </a>
</template>

<script>
export default {
  name: "",
  //引入a标签特性鼠标移入悬浮文字
  props:['title'],
 //当前this.$attrs,this.$listeners，已经获取到了父组件中所有的属性和方法
  mounted(){
    console.log(this.$attrs,this.$listeners)
  }
};
</script>
```

```
<style lang="less" scoped>
</style>
```

七、$parent / $children 与ref

概念：

```
理解：
    $children：所有直接子组件对象的数组，利用它可以更新多个子组件的数据
    $parent：父组件对象，从而可以更新父组件的数据
    $refs：包含所有有ref属性的标签对象或组件对象的容器对象
利用它们能做什么？
    能方便的得到子组件/后代组件/父组件/祖辈组件对象，从而更新其data或调用其方法
    官方建议不要大量使用，优先使用props和event
    在一些UI组件库定义高复用组件时会使用$children和$parent，如Carousel组件
扩展：
    问题：多个组件有部分相同的js代码 ？
    如何实现vue组件中js代码的复用呢？  利用vue的mixin技术实现
    什么时候使用：当多个组件的JS配置部分有一些相同重复的代码时
    本质：实现Vue组件的JS代码复用，简化编码的一种技术
```

- 子实例可以用this.$parent访问父实例
- 子实例被推入父实例的$children
- ref：如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例
- 这两种方法的弊端是，无法在跨级或兄弟间通信。
- ref 是被用来给元素或子组件注册引用信息的。引用信息将会注册在父组件的 $refs 对象上。

parent

- **类型**： `Vue instance`
- **详细**：

  指定已创建的实例之父实例，在两者之间建立父子关系。子实例可以用 `this.$parent` 访问父实例，子实例被推入父实例的 `$children` 数组中。

  节制地使用 `$parent` 和 `$children` - 它们的主要目的是作为访问组件的应急方法。更推荐用 props 和 events 实现父子组件通信

直接操作父子组件的实例。$parent 就是父组件的实例对象，而 $children 就是当前实例的直接子组件实例了，不过这个属性值是数组类型的，且并不保证顺序，也不是响应式的。

通过$parent（获取父组件对象原型对象中的参数）和$children（获取子组件对象原型对象中的参数）就可以访问组件的实例，则可以访问此组件的所有方法和data来实现组件间通信

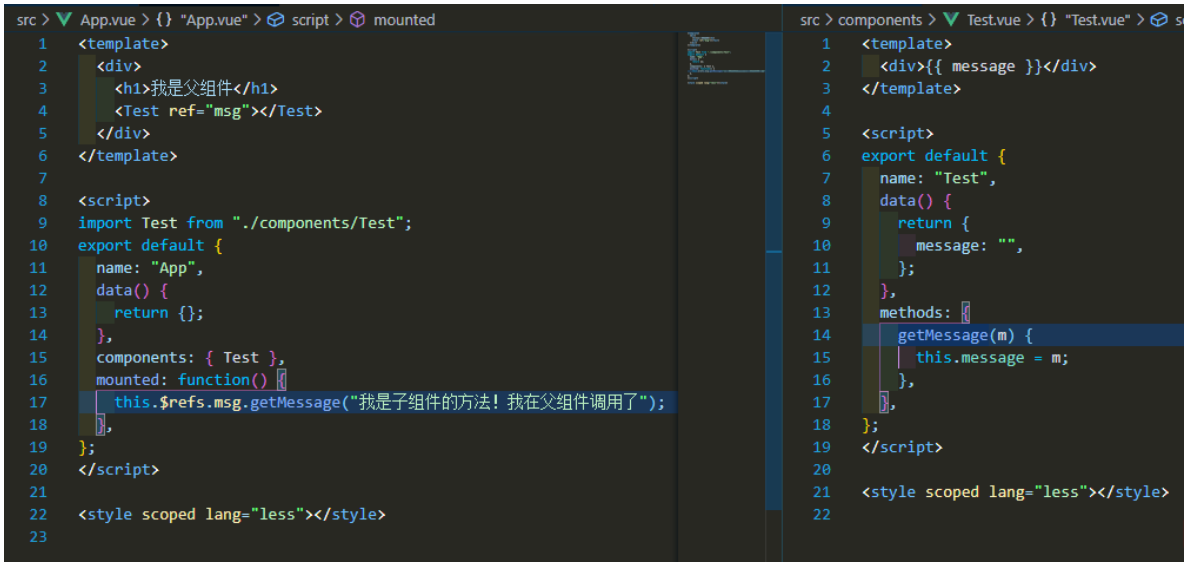
注意：

要注意边界情况，如在#app上拿$parent得到的是new Vue()的实例，在这实例上再拿$parent得到的是undefined，而在最底层的子组件拿$children是个空数组。也要注意得到$parent和$children的值不一样，$children 的值是数组，而$parent是个对象

ref访问子组件实例或子元素

简单来说，就是父组件在模板中定义了ref这个标签属性时，则可以获取子组件的属性和方法。

在父组件中就可以使用子组件的方法和属性



如果ref用在子组件上，指向的是组件实例，可以理解为对子组件的索引，通过$ref可能获取到在子组件里定义的属性和方法。

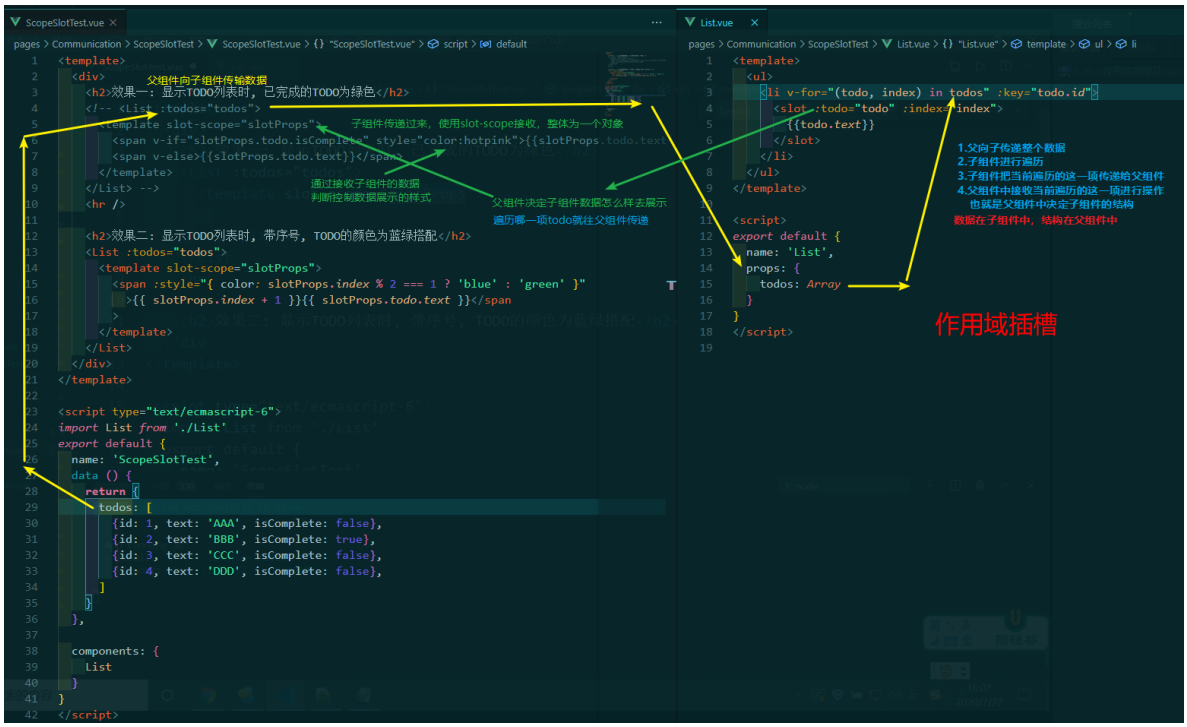
如果ref在普通的 DOM 元素上使用，引用指向的就是 DOM 元素，通过$ref可能获取到该DOM 的属性集合，轻松访问到DOM元素，作用与JQ选择器类似。

八、作用域插槽slot-scope

插槽标签：slot，标签内的数据是不定的

```
什么情况下使用作用域插槽?
    父组件需要向子组件传递标签结构内容
    但决定父组件传递怎样标签结构的数据在子组件中
编码:
    子组件:
        <slot :row="item" :$index="index">  <!-- slot的属性会自动传递给父组件 -->
        </slot>
    父组件:
        <template slot-scope="{row, $index}">
            <span>{{$index+1}}</span>   
            <span :style="{color: $index%2===1 ? 'blue' : 'green'}" >
{{row.text}}</span>
        </template>
使用作用域插槽:
    对于封装列表之类的组件特别需要
    element-ui中: Table组件中就用到了slot-scope
```

图示：



九、vuex 通信方式

概念：

vuex用来统一管理多个组件共享的状态数据
任意要进行通信的2个组件利用vuex就可以实现
A组件触发action或mutation调用，将数据保存到vuex的状态中
B组件读取vuex中的state或getters数据，得到最新保存的数据进行显示

- **vuex 是 vue 的状态（数据）管理**
- 五个核心概念：state mutations actions getters modules
  - **state**：是专门用来存放数据的地方（官方所说的状态，又叫状态数据）
  - **mutations**：是专门用来存放更新数据的各种方法的地方（这些方法，必须是直接修改数据的方法，不能存在：判断、循环、异步）
  - **actions**：专门用来存放和组件行为进行对接的各种方法的地方（可以存在：判断、循环、异步）比如异步接口请求
  - **getters**：计算数据（属性的地方）
  - **modules**：模块化Vuex

关于modules：

目录结构《以下为两个模块feeds和movies》：

```
store
|  index.js
|
├─feeds
|      actions.js
```

```
|      getters.js
|      index.js
|      mutation-type.js
|      mutations.js
|      state.js
|
└─movies
        actions.js
        getters.js
        index.js
        mutation-type.js
        mutations.js
        state.js
```

第一步：在store文件夹下的index.js入口文件写入：

```
import Vue from 'vue';
import Vuex from 'vuex';
import feeds from './feeds';
import movies from './movies';

Vue.use(Vuex);

export default new Vuex.Store({
  modules: {
    feeds,
    movies
  },
});
```

第二步：在每个模块内的index文件这组装所有的零件，并且输出：

注意下面多出的一行，我们在组件里怎么区分不同模块呢？namespaced写成true，意思就是可以用这个module名作为区分了（也就是module所在的文件夹名）

```
import state from './state';
import mutations from './mutations';
import actions from './actions';
import getters from './getters';

export default {
        namespaced: true,  //多出的一行
        state,
        mutations,
        actions,
        getters
};
```

第三步：在组件里使用：

使用的时候

- 获取state，这里使用映射：

```
import { mapState, mapMutations } from "vuex";

export default {
computed:{
    ...mapStated('模块名（嵌套层级要写清楚）',{  //比如'movies/hotMovies
        a:state=>state.a,
        b:state=>state.b
    })
},
```

- 触发actions操作:

```
import { mapActions } from 'vuex'
methods:{
    ...mapActions('模块名（嵌套层级要写清楚）',[  //比如'movies/getHotMovies
        'foo',
        'bar'
    ])
}
```

知识点补充:

- **element-ui　中的button**

- **Mixin混合技术对组件中的js代码复用解耦**